Smoothing Region Boundaries in Variable Depth Mapping for Real Time Stereoscopic Images

Nick Holliman

Department of Computer Science, University of Durham, Durham, United Kingdom

ABSTRACT

We believe the need for stereoscopic image generation methods that allow simple, high quality content creation continues to be a key problem limiting the widespread up-take of 3D displays. We present new algorithms for creating real time stereoscopic images that provide increased control to content creators over the mapping of depth from scene to displayed image.

Previously we described a Three Region, variable depth mapping, algorithm for stereoscopic image generation. This allows different regions within a scene to be represented by different ranges of perceived depth in the final image. An unresolved issue was that this approach can create a visible discontinuity for smooth objects crossing region boundaries. In this paper we describe two new Multi-Region algorithms to address this problem: boundary smoothing using additional sub-regions and scaling scene geometry to smoothly vary depth mapping.

We present real time implementations of the Three-Region and the new Multi-Region algorithms for OpenGL to demonstrate the visual appearance of the results. We discuss the applicability and performance of each approach for rendering real time stereoscopic images and propose a simple modification to the standard graphics pipeline to better support these algorithms.

Keywords: Stereoscopic Cameras, Graphics Pipeline, Human Factors, Computer Graphics, Multi-View, 3D Display

1. INTRODUCTION

The perceived depth seen in a stereoscopic image varies with a wide range of parameters and, even if the display device is ideally aligned and has zero crosstalk, it can be difficult to compose the depth presentation as desired. Partly this is due to well-known human factors considerations; there is a limit to the range of usable perceived depth before the binocular image becomes diplopic.[1–3] However, even within this range existing stereoscopic image generation methods provide limited control over the mapping between perceived depth in the displayed image and the original scene depth.

To address this problem we previously presented the Three-Region algorithm[4]; this allows different regions of the scene to be mapped to different ranges of perceived depth on a target 3D display. The benefit is a particular region of interest in the scene can be given additional depth representation in the stereoscopic image. The aim is to retain the scene outside the region of interest, rather than clip it or disguise it by blurring because in many applications it is important to see context around the region of interest. For example, in a scientific visualization a particular feature can then be highlighted in depth or in computer games a game character can be given preferred depth representation. The approach can be compared to the "focus+context" methods used in visualization[5] allowing the "focus" to be varied in the perceived depth dimension.

A potential problem with the Three-Region algorithm occurs when smooth objects cross the boundaries between two adjacent regions. If there is a significant change in perspective projection between the two regions there can be a visible discontinuity at the boundary. In this paper we specifically address this problem and consider how a Multi-Region approach can be used to smooth the visual transition at region boundaries. We present two new Multi-Region algorithms, one using linear perspective projection and an alternative using scene scaling to control perceived depth. In addition we describe versions of all three algorithms using asymmetric camera frusta so that we can implement them using real time graphics in OpenGL.[6]

Further author information: (Send correspondence to Nick Holliman.)
Email: n.s.holliman@durham.ac.uk, Telephone: +44 191 334 1761, Web: http://www.durham.ac.uk/n.s.holliman/

2. BACKGROUND

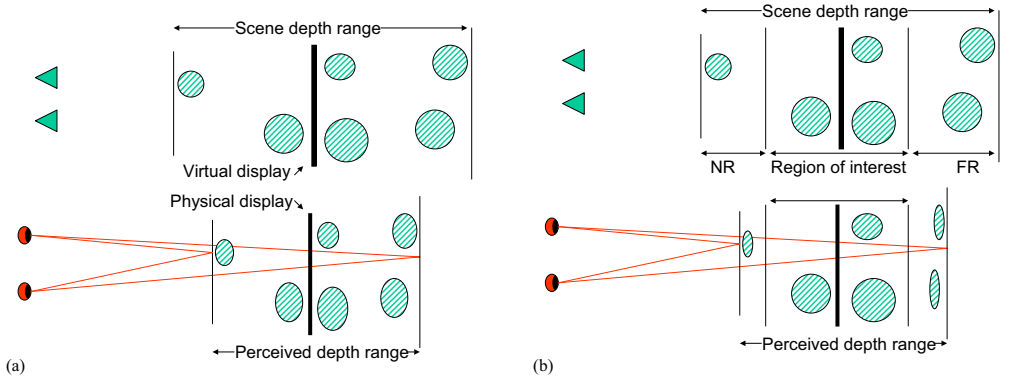


Figure 1. Stereoscopic image capture maps a scene depth range to a display perceived depth range: (a) **Single Region** algorithms map a range of scene depth to perceived depth on a target display as a whole. Some algorithms[7] guarantee the result of this mapping even when the viewers head is moving. (b) The **Three Region** algorithm[4] provides additional control allowing the available stereoscopic depth range to be split preferentially between three regions. A defined region of interest can then be given the best available stereo depth representation.

Much recent work in stereoscopic image generation[7,8] has implicitly or explicitly investigated Single Region depth mapping, taking a scene as a whole and mapping it to a target perceived depth range on a 3D display, as in figure 1(a). The most recent algorithms are able to guarantee the maximum depth seen in an image. These algorithms also provide guarantees about the depth seen in images for real time head-tracked displays and do not have the shearing and warping effects often associated with head movement and stereo images.

There has been relatively little reported work extending this to allow the limited available perceived depth to be focused on a particular region of interest within the scene. Our Three-Region algorithm,[4] illustrated in figure 1(b), implements this using different perspective camera projection to alter the depth mapping from defined ranges of scene depth to defined ranges of perceived display depth. A similar concept using scene scaling is presented in[9] however key details of the approach were not reported and no results were presented.

3. THREE-REGION ALGORITHM FOR ASYMMETRIC CAMERA FRUSTUM

The Three Region algorithm previously reported[4] was designed for use with graphics systems that support symmetric camera frusta, such as the ray tracing program POVRay. A consequence of using symmetric frustum is the need to compute image regions not required in the final stereoscopic image pair. Image cropping regions must then be calculated and applied to all the left and right partial images making up the stereoscopic image pair.

The need to calculate and apply cropping is a small overhead in the POVRay implementation where rendering times are measured in seconds or minutes. It becomes more of a disadvantage for real time systems where the extra rendering and image manipulation required will directly impact maximum achievable frame rates. Fortunately, graphics APIs such as OpenGL support asymmetric camera frustum allowing us to render only the regions of the stereoscopic images we need. We therefore derive a modified Three Region algorithm for use with graphics systems supporting asymmetric camera frustum.

Recall[4] that we consider two distinct geometries; the geometry defining the relationship between the viewer and the display and the geometry of the scene and camera. The three-region algorithm maps three defined regions, Near Region (NR), Region of Interest (ROI) and Far Region (FR) in scene depth onto corresponding defined ranges of geometric perceived depth *gpd* in the displayed image. This mapping meets the constraint that points on the region boundaries are projected to coincident positions and hence depth in the image pair, whichever region they are considered to belong to.

For each defined region we calculate, as previously, a camera separation that maps scene depth range to the target perceived depth range and use this to calculate an appropriate camera frustum as below. The notation follows that defined in our earlier paper.[4]

3.1. Region of Interest Mapping

We derive the camera parameters to generate the left and right partial images for the ROI using asymmetric camera frustum. The distance to the location in the scene of the zero disparity plane (ZDP), z', the width of the frustum w' at that point and the camera separation, a' are calculated as previously reported.[4] We additionally assume the height of the frustum at the ZDP, h', is simply w' scaled by the aspect ratio of the intended final image.

We need the six extents for each frustum,[6] the $left'$, $right'$, $bottom'$, top', $near'$ and far' limits, plus the cameras $from'$, to' and up' vectors. In the following we assume a stereoscopic camera with a look from point displaced about the origin looking in the direction of the positive z-axis.

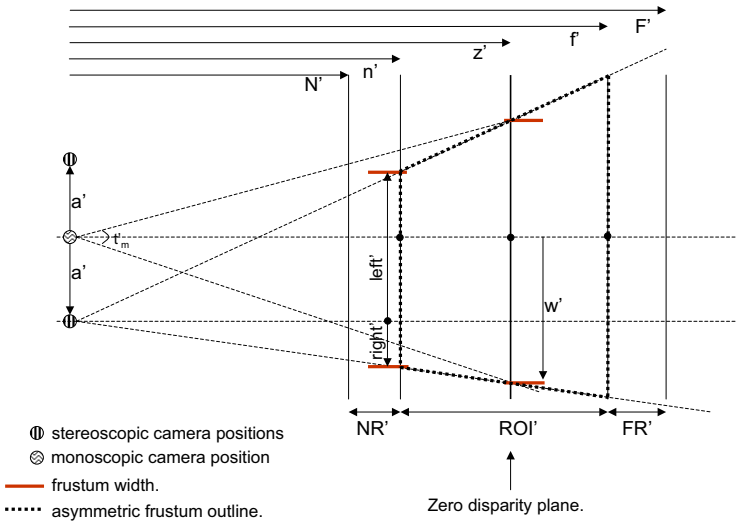


Figure 2. The scene geometry highlighting the asymmetric camera frustum for the right camera capturing the ROI.

Referring to figure 2 we find the extents for the ROI right camera frustum. We find the magnitude of the left and right extents by similar triangles using the known frustum width w' at the ZDP.

$$left' = -\frac{n'}{z'}(w' + a') \qquad right' = \frac{n'}{z'}(w' - a') \tag{1}$$

The $bottom'$ and top' extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h', while the $near'$ and far' extents of the frustum are simply the given extents of the ROI:

$$bottom' = top' = \frac{n'}{z'}(h') \qquad near' = n' \qquad far' = f' \tag{2}$$

Finally the ROI right camera location and orientation are:

$$from' = [-a', 0, 0] \quad to' = [-a', 0, z'] \quad up' = [0, 1, 0] \tag{3}$$

The frustum for the ROI left camera can be found by symmetry.

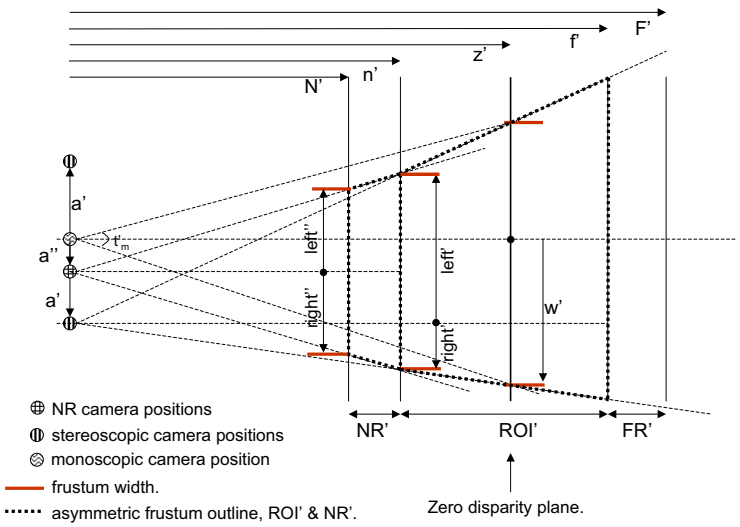


Figure 3. The scene geometry highlighting the asymmetric camera frustum for the right camera capturing the Near-Region.

3.2. Near Region Mapping

The calculation for the stereo camera separation for the near region, a'' remains as previously reported.[4]

To find the camera frustum parameters to generate the left and right partial images for the NR consider the geometry of figure 2. By inspection we know the NR frustum width at, n', as this is where the NR meets the ROI, as shown in figure 3 . To find the $left''$ and $right''$ extents for the NR frustum we simply project the ROI results $left'$ and $right'$ onto the N' plane, adjusting for the change in camera separation. Hence the magnitude of the left and right extents for the Near Region right camera frustum, $left''$ and $right''$ are given by:

$$left'' = -\frac{N'}{n'}(|left'| - a' + a'') \qquad right'' = \frac{N'}{n'}(|right'| + a' - a'') \tag{4}$$

The $bottom''$ and top'' extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h' projected to the N' plane. The $near''$ and far'' extents of the frustum are simply the given extents of the near region:

$$bottom'' = top'' = \frac{N'}{z'}(h') \quad near'' = N' \quad far'' = n' \tag{5}$$

Finally the Near Region right camera location and orientation are:

$$from'' = [-a'', 0, 0] \quad to'' = [-a'', 0, (N' + n')/2] \quad up'' = [0, 1, 0] \tag{6}$$

The Near Region left camera frustum and location can be found by symmetry.

3.3. Far Region Mapping

The Far Region frustum is derived in a similar manner to the Near Region. The magnitude of the left and right extents for the Far Region right camera frustum, $left'''$ and $right'''$, are found by projecting the known frustum width, w', at z' adjusted by the camera separation onto the plane at f' and are given by:

$$left''' = -\left(\frac{f'}{z'}(w' + a')\right) - a' + a''' \qquad right''' = \left(\frac{f'}{z'}(w' - a')\right) + a' - a''' \tag{7}$$

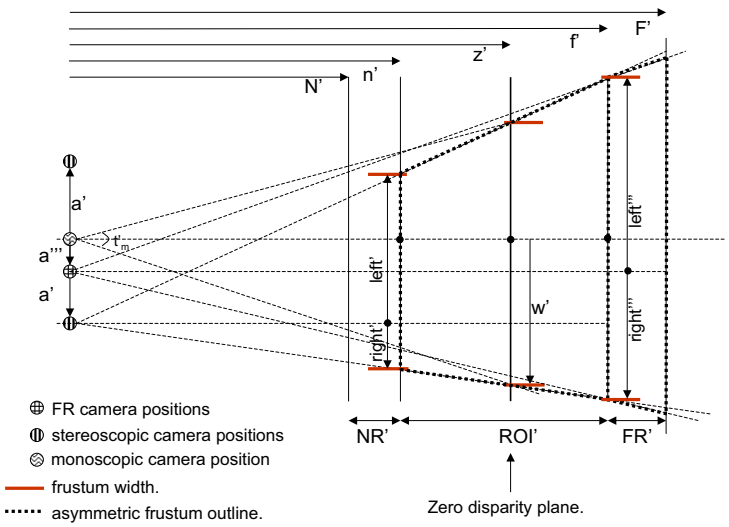


Figure 4. The scene geometry highlighting the asymmetric camera frustum for the right camera capturing the Far Region.

The $bottom'''$ and top''' extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h' projected to the f' plane. The $near'''$ and far''' extents of the frustum are simply the given extents of the far region.

$$bottom''' = top''' = \frac{f'}{z'}(h') \quad near''' = f' \quad far''' = F' \tag{8}$$

Finally the Far Region right camera location and orientation are:

$$from''' = [-a''', 0, 0] \quad to''' = [-a''', 0, (f' + F')/2] \quad up''' = [0, 1, 0] \tag{9}$$

The Far Region left camera frustum and location can be found by symmetry.

3.4. Implementation

The Three-Region algorithm described above can now be implemented in OpenGL. The left and right images in each stereo pair are formed by rendering three partial images of the three regions. The partial images are rendered in far to near order so that as they overwrite each other and the occlusions in the final image pair are correctly accounted for.

4. MULTI-REGION ALGORITHM FOR ASYMMETRIC CAMERA FRUSTUM

We anticipated that a drawback of the Three-Region algorithm in certain scenes is that would introduce a C1 discontinuity at the region boundaries. That is, while we guarantee the surface is continuous across boundaries, the surface normals are not. In scenes which have smooth objects crossing a region boundary this can result in a visible artifact; a kink can be seen as the object crosses a region boundary.

In addition when there is a significant difference in depth compression between two adjacent regions a locally high disparity gradient can be introduced into the stereoscopic image where none previously existed. It is known[10] that the effect of this can be to destroy the stereo effect in the area local to the high gradient and this is visually distracting. A possible solution is to introduce a more gradual change in perspective projection at region boundaries. Ideally, this should be continuous however within the constraints of current real time graphics pipelines we need to find a solution using linear perspective projection.

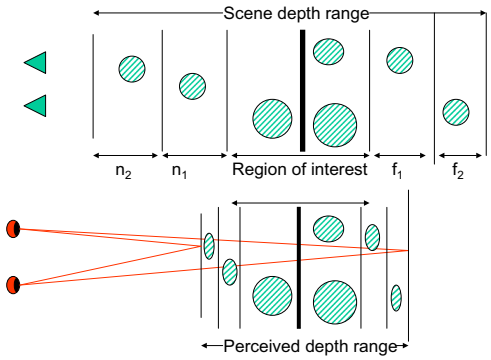


Figure 5. The Multi Region algorithm maps multiple scene regions to corresponding perceived depth regions.

The solution we describe below is the Multi-Region algorithm for asymmetric camera frustum. This extends the Three-Region approach enabling multiple additional regions to be inserted as required to smooth the transition between regions, see figure 5. We follow the same approach, with a Region of Interest surrounded by multiple near and far regions of the scene however the method is general and any number of key regions in the scene may be defined.

The input to the new algorithm is an array of scene depth boundaries and a corresponding array of display perceived depth boundaries. We then need to calculate the camera frustum and separation that maps each scene region to the available *gpd* defined for the display. The output is a set of arrays, holding for each region: the left and right camera frustum, look from point, look to point and up vector. We first consider the ROI mapping.

4.1. Region of Interest Mapping

We assume the ROI is the central region with a number of near and far regions to either side.

If there are $nRegions$ mapped from scene to display there will by definition $nRegions+1$ boundaries defining them. We use a single shared subscript index, i, for identifying both regions and region boundaries. This is initially set to the index number for the ROI region, $i = roiIndex$.

The region boundaries for the current region i are then:

$$n_i = displayBoundaries_i \quad f_i = displayBoundaries_{i+1} \quad n'_i = sceneBoundaries_i \quad f'_i = sceneBoundaries_{i+1} \tag{10}$$

The distance to the location in the scene of the zero disparity plane (ZDP), z', the width of the frustum w' at that point and the camera separation, now a'_i, are calculated as previously reported.[4] We additionally assume the height of the frustum at the ZDP, h', is simply w' scaled by the aspect ratio of the intended final image.

As described in the previous section for the Three-Region algorithm we find the left and right extents for the ROI right camera frustum.

$$rightCamLeft'_i = -\frac{n'_i}{z'}(w' + a'_i) \qquad rightCamRight'_i = \frac{n'_i}{z'}(w' - a'_i) \tag{11}$$

The $bottom'$ and top' extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h'. The $near'$ and far' extents of the frustum are simply the given extents of the ROI.

$$rightCamBottom'_i = rightCamTop'_i = \frac{n'}{z'}(h') \quad rightCamNear'_i = n' \quad rightCamFar'_i = f' \tag{12}$$

Finally the ROI right camera location and orientation are:

$$rightCamFrom'_i = [-a', 0, 0] \quad rightCamTo'_i = [-a', 0, z'] \quad rightCamUp'_i = [0, 1, 0] \tag{13}$$

The frustum for the ROI left camera can be found by symmetry. Note we retain a record of the camera separations for each region as we calculate them as they are reused in calculations for subsequent regions.

4.2. Mapping for all Near regions

The near regions are those closer to the camera than the region containing the ZDP. We can determine the frustum and camera location for each of the Near Regions iteratively working from furthest to nearest.

For each iteration of i from $i = roiIndex - 1$ to $i = 0$ repeat the following:

For easy comparision with previous definitions we rename the region boundaries for the current region, i, as:

$$N_i = displayBoundaries_i \quad n_i = displayBoundaries_{i+1} \quad N'_i = sceneBoundaries_i \quad n'_i = sceneBoundaries_{i+1} \tag{14}$$

We first calculate the camera separation for the current Near Region, a'_i using the method previously reported.[4]

We can then find the magnitude of the left extent for the Near Region right camera frustum, $rightCamLeft'_i$, using our knowledge of the left extent of the adjacent camera frustum $rightCamLeft'_{i+1}$ and the adjacent camera separation a'_{i+1}. Similarly the right extent can be found:

$$rightCamLeft'_i = -\frac{N'_i}{n'_i}(|rightCamLeft'_{i+1}| - a'_{i+1} + a'_i) \quad rightCamRight'_i = \frac{N'_i}{n'_i}(|rightCamRight'_{i+1}| + a'_{i+1} - a'_i) \tag{15}$$

The $rightCamBottom'_i$ and $rightCamTop'_i$ extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h' projected to the N'_i plane. The $rightCamNear'_i$ and $rightCamFar'_i$ extents of the frustum are simply the given extents of the current near region.

$$rightCamBottom'_i = rightCamTop'_i = \frac{N'_i}{z'}(h') \quad rightCamNear'_i = N'_i \quad rightCamFar'_i = n'_i \tag{16}$$

Finally the current Near Region right camera location and orientation are:

$$rightCamFrom'_i = [-a'_i, 0, 0] \quad rightCamTo'_i = [-a'_i, 0, (N'_i + n'_i)/2] \quad rightCamUp'_i = [0, 1, 0] \tag{17}$$

The frustum and camera parameters for the left camera for the current Near Region i can be found by symmetry.

Once we have repeated this iteration over all the Near Regions their frustum and camera parameters will have been calculated and stored with the ROI frustum for later use.

4.3. Mapping for all Far regions

The Far Regions are those further from the camera than the region containing the ZDP. We can determine the frustum and camera location for each of the Far Regions iteratively working from furthest to nearest.

For each iteration of i from $i = roiIndex + 1$ to $i = nRegions$ we repeat the following:

For easy comparison with previous definitions we rename the region boundaries for the current region, i, as:

$$f_i = displayBoundaries_i \quad F_i = displayBoundaries_{i+1} \quad f'_i = sceneBoundaries_i \quad F'_i = sceneBoundaries_{i+1} \tag{18}$$

We then calculate the camera separation for the current Far Region, a'_i using the method previously reported.[4]

We can then find the magnitude of the left extent for the Far Region right camera frustum, $rightCamLeft'_i$, using our knowledge of the left extent of the adjacent camera frustum $rightCamLeft'_{i-1}$ and the adjacent camera separation a'_{i-1}.

$$rightCamLeft'_i = -\left(\frac{f'_i * rightCamLeft'_{i-1}}{sceneBoundaries'_{i-1}}\right) - a'_{i-1} + a'_i \tag{19}$$

Similarly the right extent can be found.

$$rightCamRight'_i = \left(\frac{f'_i * rightCamRight'_{i-1}}{sceneBoundaries'_{i-1}}\right) + a'_{i-1} - a'_i \tag{20}$$

The $rightCamBottom'_i$ and $rightCamTop'_i$ extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h' projected to the f' plane. The $rightCamNear'_i$ and $rightCamFar'_i$ extents of the frustum are simply the given extents of the far region.

$$rightCamBottom'_i = rightCamTop'_i = \frac{f'_i}{z'}(h') \quad rightCamNear'_i = f'_i \quad rightCamFar'_i = F'_i \tag{21}$$

Finally the Far Region right camera location and orientation are:

$$rightCamFrom'_i = [-a'_i, 0, 0] \quad rightCamTo'_i = [-a'_i, 0, (f'_i + F'_i)/2] \quad rightCamUp'_i = [0, 1, 0] \tag{22}$$

Each of the Far Region left camera frustum and locations can be found by symmetry.

5. MULTI-REGION ALGORITHM USING SCENE SCALING

Previous work[9] has discussed using scene scaling instead of perspective projection to control the perceived depth seen on the display. The idea is to scale the scene to fit defined depth boundaries before it is projected. The results can be anticipated to be a scene compressed in depth with a significantly altered perspective depth compared to the original monoscopic camera view, or the Multi-Region algorithm.

We derive the Multi-Region algorithm using scene scaling below. The algorithm operates in a similar manner to the Multi-Region algorithm for perspective projection described above. The scene is clipped into ranges but rather than using a different perspective projection for each region the regions are scaled by a pre-calculated factor using the OpenGL ModelView transformation matrix. For clarity we describe an algorithm that assumes unit vertical and horizontal scaling and we are simply concerned with the depth scaling from scene to display, these can be easily calculated if required.

5.1. Mapping for All Regions

In this algorithm the mapping for all $nRegions$ can be calculated identically as the camera position and field of view remain constant for all regions. The parameters that change for each region are the camera frustum, and the scaling from scene to display space.

We start by initialising parameters for the ROI region, setting $i = roiIndex$. For convenience we define the following:

$$n_i = displayBoundaries_i \quad f_i = displayBoundaries_{i+1} \quad n'_i = sceneBoundaries_i \quad f'_i = sceneBoundaries_{i+1} \tag{23}$$

The ZDP can then be assumed to be at the center of the ROI region in display space:

$$z' = (f_i + n_i)/2.0; \tag{24}$$

Given the monoscopic camera field of view, t'_m the width and the height of the image plane at the ZDP are then:

$$w' = z' * tan(t'_m/2.0) \quad h' = w' * 3.0/4.0 \tag{25}$$

As we have assumed unit scaling between scene and display, except in depth, then the camera separation for all regions is the eye separation e:

$$a' = e \tag{26}$$

Now we can find the frustum for each region of the scene from nearest to furthest.

For each iteration of i from $i = 0$ to $i = nRegions - 1$ we repeat the following:

Let the region boundaries for the current region i be:

$$n_i = displayBoundaries_i \quad f_i = displayBoundaries_{i+1} \quad n'_i = sceneBoundaries_i \quad f'_i = sceneBoundaries_{i+1} \tag{27}$$

Then the scaling we need to apply to the current region so that it maps into the available perceived depth range is:

$$scaling_i = (f_i - n_i)/(f'_i - n'_i) \tag{28}$$

The left and right frustum extents for the right camera are given by the following:

$$rightCamLeft'_i = -\frac{n_i}{z'}(w' + a') \quad rightCamRight'_i = \frac{n_i}{z'}(w' - a') \tag{29}$$

The bottom and top extents are equal as the frustum is vertically symmetrical and can be found using similar triangles and the height of the frustum h'. The near and far extents of the frustum are the given extents of the current region:

$$rightCamBottom'_i = rightCamTop'_i = \frac{n_i}{z'}(h') \quad rightCamNear'_i = n' \quad rightCamFar'_i = f' \tag{30}$$

Finally the right camera location and orientation are:

$$rightCamFrom'_i = [-a', 0, 0] \quad rightCamTo'_i = [-a', 0, z'] \quad rightCamUp'_i = [0, 1, 0] \tag{31}$$

The left camera frustum and locations can be found by symmetry.

5.2. Implementation

A side effect of scaling the scene before rendering is also to scale the surface normals used for shading. These can be partially corrected by re-normalizing them to unit length using OpenGL extensions. Ideally the surface normals need re-calculating each time the region scaling changes and OpenGL can support this.

6. REAL TIME IMPLEMENTATION

Four image generation algorithms have been implemented in OpenGL using asymmetric frusta; Single Region,[7] Three Region,[4] Multi Region using perspective and Multi Region using scaling. This allows a direct comparison of results obtained from applying each method to the same test scene.

The Three and Multi Region algorithms were implemented using multi-pass rendering, rendering each frustum in turn from far to near. For example, to create the final image for the right channel in the Multi Region algorithm with $nRegions = 5$ requires five partial image renderings, overlaid one on another. The partial images for the left and right views are illustrated in figure 6.

The scaling of each partial image from its frustum to the render window is handled transparently as the render window size in pixels is kept the same for each partial rendering. No cropping is required because of the use of the asymmetric frusta.

The algorithms were implemented and tested using a 3GHz Pentium PC with an nVidia Quadro FX 4000 dual DVI graphics card. The PC drives a BARCO Gemini display; two passive circularly polarised projectors, with a 2.4m PASCAD low-crosstalk screen. A benefit of the FX4000 is the driver's ability to handle a wide range of display types via the standard OpenGL quad-buffer mechanism; hence we were also able to view the results using a Sharp LL151 desktop display and a SeeReal C-i display. An implementation for demonstration on the Sharp RD3D laptop display used the Sharp interlacing libraries in place of the nVidia drivers.

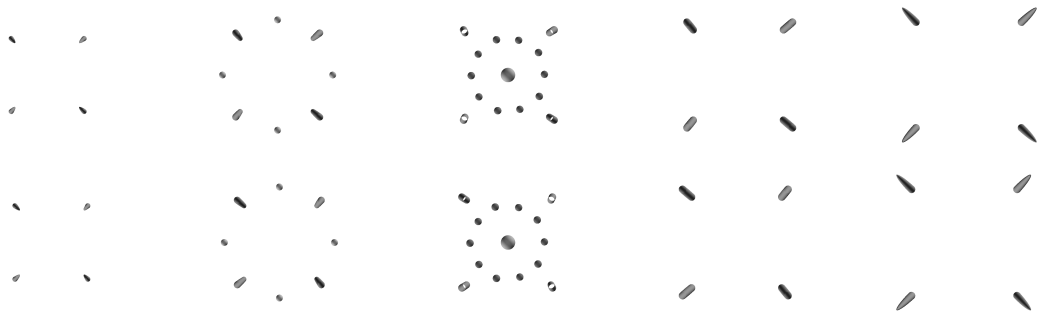


Figure 6. The partial images that make up the Multi-Region rendering using five sub-regions. The upper row shows the left images and the lower row the right images.

7. RESULTS

The test scene consists of four, long, thin ellipsoids stretching from the furthest to nearest scene depth limits, four far spheres, a central larger sphere and a central ring of smaller spheres with a pattern of alternate depth differences. The central ring of spheres acts as the region of interest and we would expect the pattern of depth differences in the ring of spheres to be better represented when rendered with the Three and Multi Region algorithms compared to the Single Region algorithm. Additionally we would anticipate the long ellipsoids to have visible artifacts at region boundaries if rendered using the Three Region algorithm and less visible artifacts using the Multi Region algorithms. The results can be seen by free fusing the image pairs in figure 7, these images can also be downloaded from the author's web site.

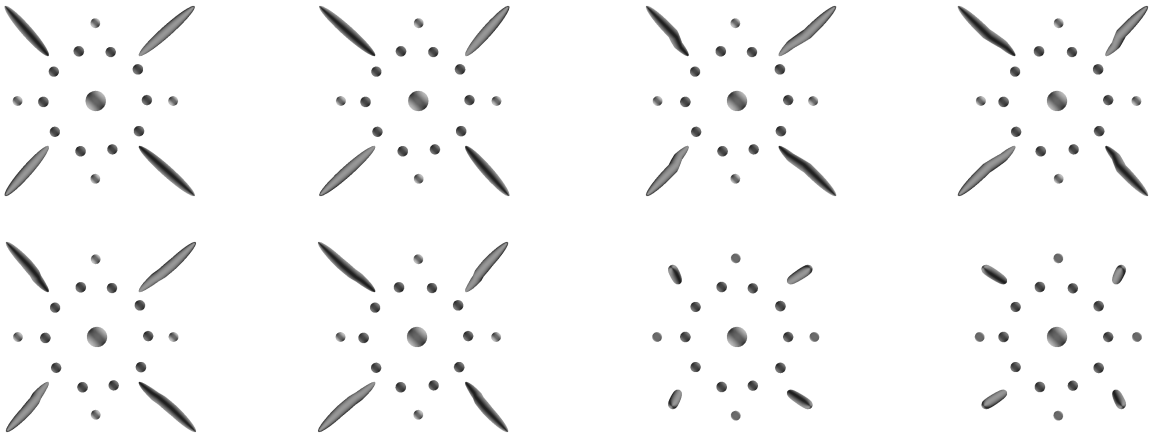
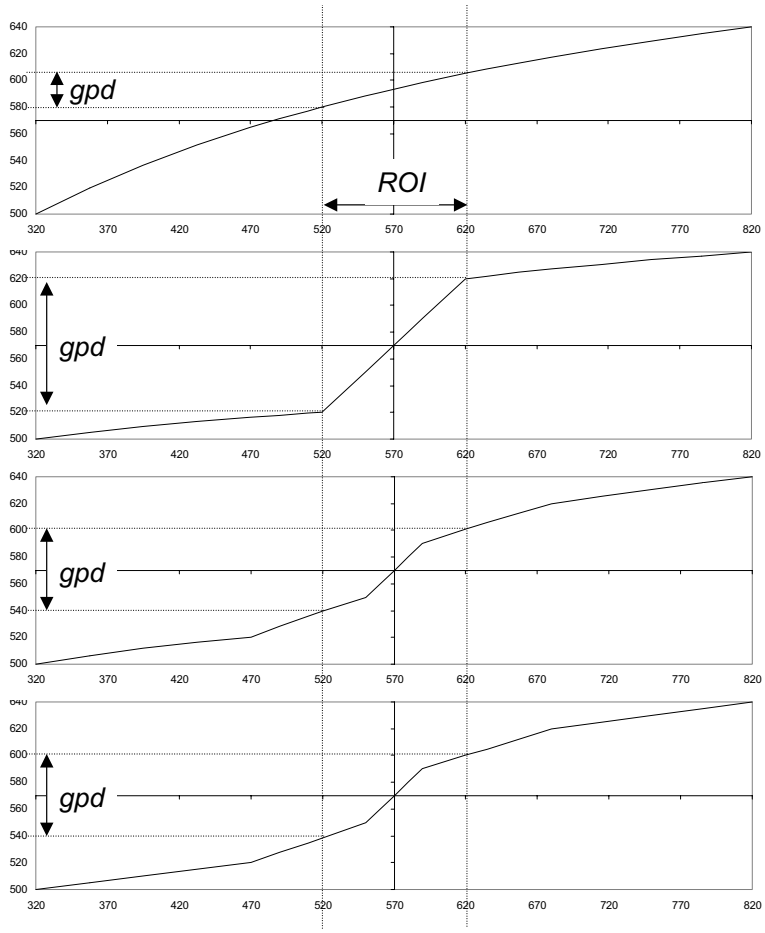


Figure 7. The results of applying four different depth mapping algorithms to the same test scene. Top left is the Single Region algorithm. Top right is the Three Region algorithm. Bottom left is the Multi Region algorithm using perspective projection. Bottom right is the Multi Region algorithm using scene scaling.

As anticipated depth in the region of interest is reproduced well enough to clearly distinguish the depth differences in the ring of spheres using the Three and Multi Region algorithms whereas these differences are significantly less clear using the Single Region algorithm. In addition there are shading and disparity artifacts at region boundaries in the projection of the long ellipsoids using the Three Region algorithm and these are reduced by the Multi Region algorithm using perspective.

The Multi Region algorithm using scaling produces a quite different projection of the scene having scaled the scene geometry before projection. This significantly changes the image of the scene and it is application dependent as to whether it is preferable to the other approaches.



a) Single-region depth mapping
The image creator only controls the end points of the mapping. The *ROI* receives little of the available perceived depth *(gpd)* in the displayed image.

b) Three-region depth mapping
The image creator has control of the boundaries of each region. The *ROI* can be allocated most of the perceived depth *(gpd)* in the displayed image.

c) Multi-region using perspective
Here the additional regions are used to smooth the rate of change of disparity at the boundaries of the *ROI*.

d) Multi-region using scaling
While this appears to have a similar mapping to *c)* the visual effect over the whole image is significantly different.

Figure 8. The depth transfer functions, from scene to display, for four stereoscopic image generation algorithms applied to the test scene. Scene depth of $[320, 820]$ along the horizontal axis is mapped onto perceived depth of $[500, 640]$ along the vertical axis. The two axes cross at the point defined by the middle of the scene and the display plane. Note, in case a) this is not the ZDP which is shifted by the mapping outside the ROI.

The graphs in figure 8 illustrate the depth mapping from the scene to the displayed image for each of the four algorithms. These are plots along a line from the center of the viewers eye position through the center of the displayed image.

8. DISCUSSION

We anticipated the Three Region algorithm would have problems for the long thin objects oriented towards the camera and that at region boundaries a visible shading artifact would be present. This turns out to be the case and these are visible in the images illustrated in figure 7. The new Multi Region algorithm using perspective does help overcome the problem by smoothing the transition in depth from the ROI to the adjacent regions. This difference can also be seen comparing the graphs in figure 8 b) and c).

A second effect the Multi Region algorithm helps overcome is the steep disparity gradient the Three Region algorithm can introduce near region boundaries, as shown in figure 8 b). Such steep disparity gradient is known to have significant local affect on Panum's Area.[10] By smoothing the transition between regions the Multi Region algorithm can be used to reduce the local disparity gradient to subjectively acceptable levels.

For the scene scaling Multi Region algorithm the scene is scaled using the ModelView transform avoiding adding additional load to the CPU. However, the normals must be re-generated, as the shape of the geometry has

been changed, before the vertex lighting calculations have been performed. This requires additional computation compared to the Multi Region algorithm using perspective.

Overall, additional computational overhead is required by all Three and Multi Region algorithms. Each stereo image pair requires *nRegions* rendering passes and although geometry is clipped so that only relevant geometry is rendered all the scene must be passed from CPU to GPU for each rendering pass. For static scenes retained display lists could help reduce this overhead by storing the geometry locally in the graphics card.

For improved performance it would be possible to parallelize the computation of the partial images using multiple pipelines with a shared z-buffer to resolve the depth order in the final composed image. A better solution would be to modify the graphics pipeline to allow it to store multiple projection matrices and choose which to apply based on individual vertex depth. The scene would then only need to be sent from CPU to GPU once and frame rates could be maintained for all algorithms at limited additional expense in pipeline complexity.

9. CONCLUSIONS

We have considered four algorithms for drawing stereoscopic images; two new algorithms, Multi Region using perspective and Multi Region using scaling and two existing algorithms, Single Region and Three Region. We derived versions of the algorithms for asymmetric camera frustum allowing us to implement them using OpenGL real time graphics on a range of 3D displays.

Our aim in developing the new Multi Region algorithm using perspective was to remove visual artifacts we saw at region boundaries when using the existing Three Region algorithm for rendering long thin objects oriented towards the camera. The new algorithms have demonstrated two benefits; a reduction in the visible shading artifact at region boundaries and smoothing the local disparity gradient at region boundaries allowing us to retain a deep Panum's Area and avoid loss of fusion. The additional computational cost of using the Multi Region algorithm is not prohibitive and we have suggested simple changes to the real time graphics pipeline that would make the overhead minimal.

We investigated a version of the Multi Region algorithm using scene scaling to control depth mapping as this had been an approach suggested in the literature.[9] However, it is not clear what practical benefit this algorithm has as it requires additional work to correct vertex normals and significantly alters the perspective projection of the scene. It may also require adjustments to texture coordinates.

In conclusion, we believe the new Multi Region algorithm using perspective has benefits for the content creator allowing improved control over the depth composition in stereoscopic images and improved stereoscopic image quality compared to existing methods. The algorithm allows a region of interest to be highlighted in depth providing stereoscopic "focus+context" for interacting with complex scenes.

REFERENCES

1. A. Woods, T. Docherty, and R. Koch, "Image distortions in stereoscopic video systems," *Proceedings of SPIE* **1915**, 1993.
2. Y. Yeh and L. Silverstein, "Limits of fusion and depth judgements in stereoscopic color displays," *Human Factors* **1**(32), 1990.
3. N. Holliman, "3D Display Systems." to appear; Handbook of Opto-electronics, IOP Press, Spring 2005, ISBN 0-7503-0646-7.
4. N. Holliman, "Mapping perceived depth to regions of interest in stereoscopic images," in *Stereoscopic Displays and Virtual Reality Systems XI*, *Proceedings of SPIE* **5291**, 2004.
5. C. Ware, *Information visualization, perception for design*, Morgan Kaufmann, 1999. ISBN 0-1-55860-511-8.
6. D. Schreiner, M. Woo, J. Neider, and T. Davis, eds., *OpenGL Programming Guide, Version 1.4, Fourth Edition*, Addison-Wesley Pub Co, 1993. ISBN 0321173481.
7. G. Jones, D. Lee, N. Holliman, and D. Ezra, "Controlling perceived depth in stereoscopic images," in *Stereoscopic Displays and Virtual Reality Systems VIII*, *Proceedings of SPIE* **4297A**, 2001.
8. Z. Wartell, *Stereoscopic Head-Tracked Displays: Analysis and Development of Display Algorithms*. PhD thesis, Georgia Institute of Technology, 2001.
9. S. Williams and R. Parrish, "New computational control techniques and increased understanding for stereo 3D displays," *Proceedings of SPIE* **1256**, 1990.
10. P. Burt and B. Julesz, "A disparity gradient limit for binocular fusion," *Perception* **9**, 1980.